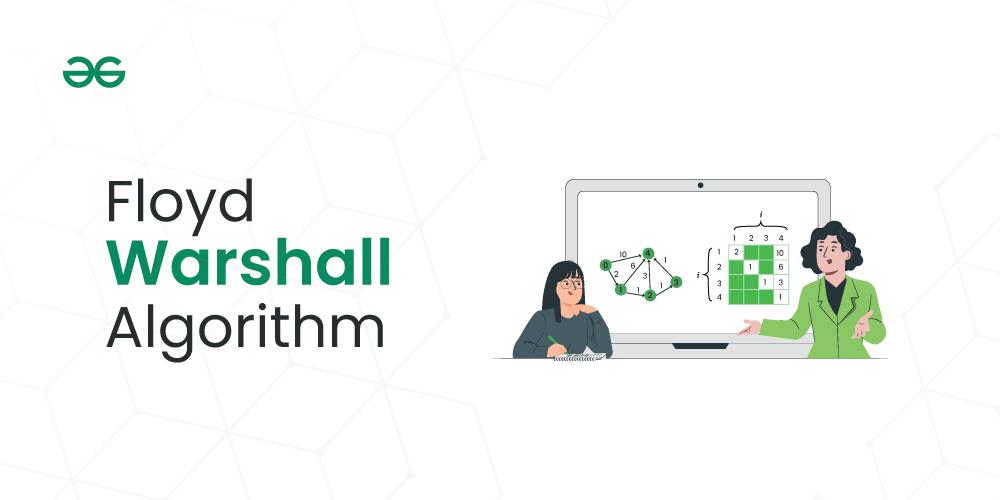
# **Floyd Warshall Algorithm**

The **Floyd-Warshall algorithm**, named after its creators **Robert Floyd and Stephen Warshall**, is a fundamental algorithm in computer science and graph theory. It is used to find the shortest paths between all pairs of nodes in a weighted graph. This algorithm is highly efficient and can handle graphs with both **positive** and n**egative edge weights**, making it a versatile tool for solving a wide range of network and connectivity problems.

* [Floyd-Warshall Algorithm](https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/#real-world-applications-of-floydwarshall-algorithm)



## **Floyd Warshall Algorithm:**

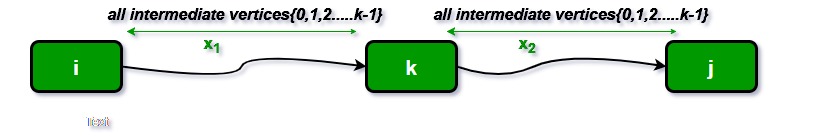
*The* ***Floyd Warshall Algorithm*** *is an all pair shortest path algorithm unlike* [*Dijkstra*](https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/) *and* [*Bellman Ford*](https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/) *which are single source shortest path algorithms. This algorithm works for both the* ***directed*** *and* ***undirected weighted*** *graphs. But, it does not work for the graphs with negative cycles (where the sum of the edges in a cycle is negative). It follows* [*Dynamic Programming*](https://www.geeksforgeeks.org/introduction-to-dynamic-programming-data-structures-and-algorithm-tutorials/) *approach to check every possible path going via every possible node in order to calculate shortest distance between every pair of nodes.*

## **Idea Behind Floyd Warshall Algorithm:**

*Suppose we have a graph* ***G[][]*** *with* ***V*** *vertices from* ***1*** *to* ***N****. Now we have to evaluate a* ***shortestPathMatrix[][]*** *where s****hortestPathMatrix[i][j]*** *represents the shortest path between vertices* ***i*** *and* ***j****.*

*Obviously the shortest path between* ***i*** *to* ***j*** *will have some* ***k*** *number of intermediate nodes. The idea behind floyd warshall algorithm is to treat each and every vertex from* ***1*** *to* ***N*** *as an intermediate node one by one.*

*The following figure shows the above optimal substructure property in floyd warshall algorithm:*

**

## **Floyd Warshall Algorithm Algorithm:**

* Initialize the solution matrix same as the input graph matrix as a first step.
* Then update the solution matrix by considering all vertices as an intermediate vertex.
* The idea is to pick all vertices one by one and updates all shortest paths which include the picked vertex as an intermediate vertex in the shortest path.
* When we pick vertex number **k** as an intermediate vertex, we already have considered vertices **{0, 1, 2, .. k-1}** as intermediate vertices.
* For every pair **(i, j)** of the source and destination vertices respectively, there are two possible cases.
  + **k** is not an intermediate vertex in shortest path from **i** to **j**. We keep the value of **dist[i][j]** as it is.
  + **k** is an intermediate vertex in shortest path from **i** to **j**. We update the value of **dist[i][j]** as **dist[i][k] + dist[k][j],** if **dist[i][j] > dist[i][k] + dist[k][j]**

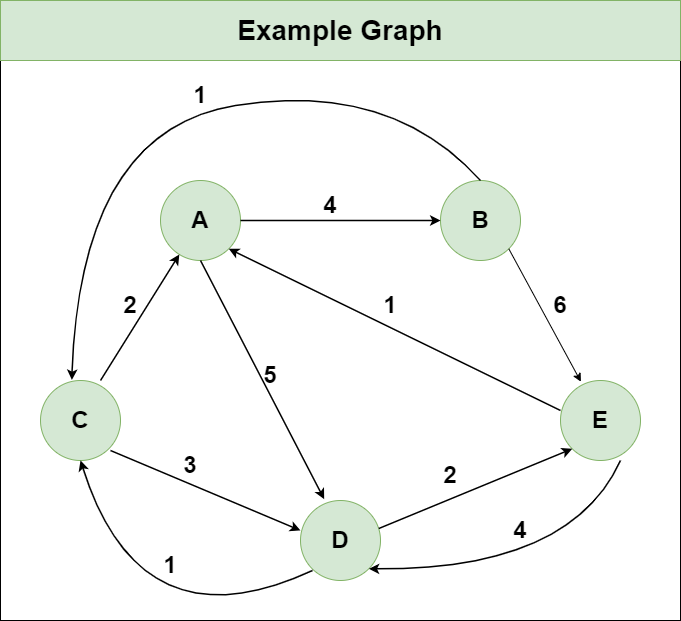
## **Pseudo-Code of Floyd Warshall Algorithm :**

*For k = 0 to n – 1  
For i = 0 to n – 1  
For j = 0 to n – 1  
Distance[i, j] = min(Distance[i, j], Distance[i, k] + Distance[k, j])*

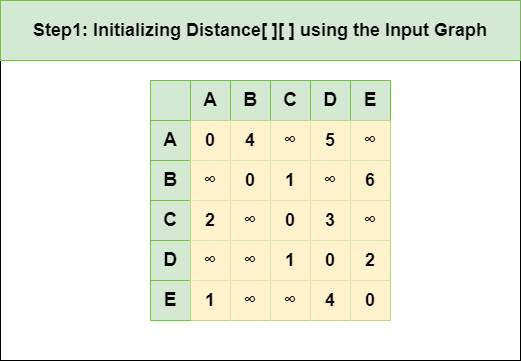
*where i = source Node, j = Destination Node, k = Intermediate Node*

## **Illustration of Floyd Warshall Algorithm :**

*Suppose we have a graph as shown in the image:*

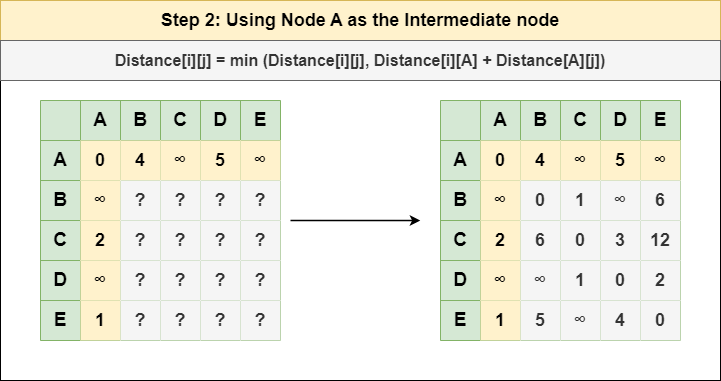
**

***Step 1:*** *Initialize the Distance[][] matrix using the input graph such that Distance[i][j]= weight of edge from* ***i*** *to* ***j****, also Distance[i][j] = Infinity if there is no edge from* ***i*** *to* ***j.***

******

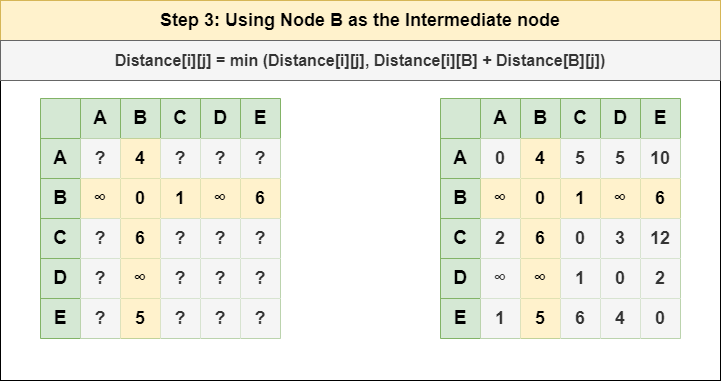
***Step 2****: Treat node* ***A*** *as an intermediate node and calculate the Distance[][] for every {i,j} node pair using the formula:*

*= Distance[i][j] = minimum (Distance[i][j], (Distance from i to* ***A****) + (Distance from* ***A*** *to j ))  
= Distance[i][j] = minimum (Distance[i][j], Distance[i][****A****] + Distance[****A****][j])*

**

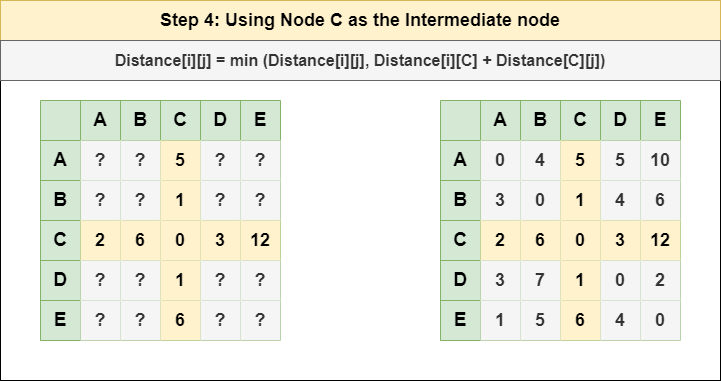
***Step 3****: Treat node* ***B*** *as an intermediate node and calculate the Distance[][] for every {i,j} node pair using the formula:*

*= Distance[i][j] = minimum (Distance[i][j], (Distance from i to* ***B****) + (Distance from* ***B*** *to j ))  
= Distance[i][j] = minimum (Distance[i][j], Distance[i][****B****] + Distance[****B****][j])*

**

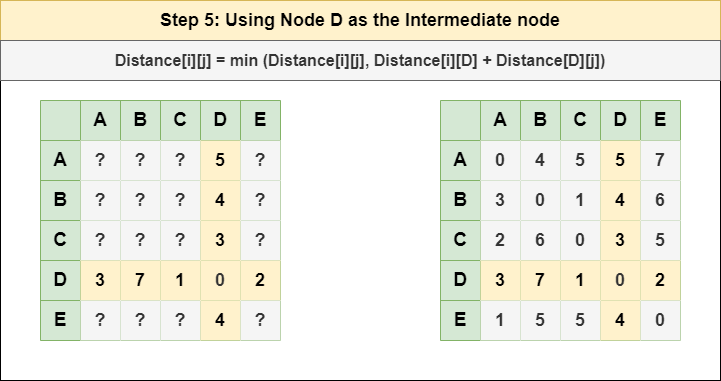
***Step 4****: Treat node* ***C*** *as an intermediate node and calculate the Distance[][] for every {i,j} node pair using the formula:*

*= Distance[i][j] = minimum (Distance[i][j], (Distance from i to* ***C****) + (Distance from* ***C*** *to j ))  
= Distance[i][j] = minimum (Distance[i][j], Distance[i][****C****] + Distance[****C****][j])*

**

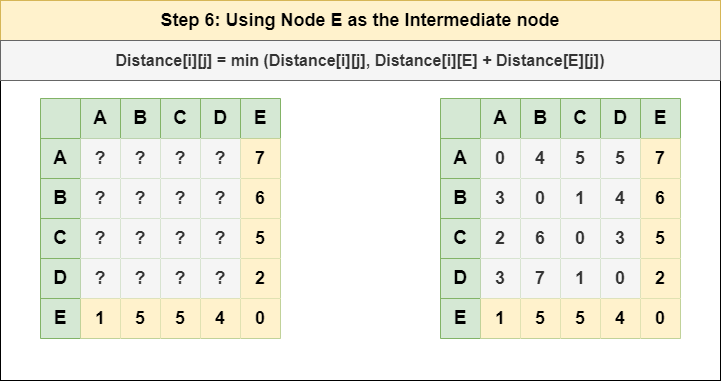
***Step 5****: Treat node* ***D*** *as an intermediate node and calculate the Distance[][] for every {i,j} node pair using the formula:*

*= Distance[i][j] = minimum (Distance[i][j], (Distance from i to* ***D****) + (Distance from* ***D*** *to j ))  
= Distance[i][j] = minimum (Distance[i][j], Distance[i][****D****] + Distance[****D****][j])*

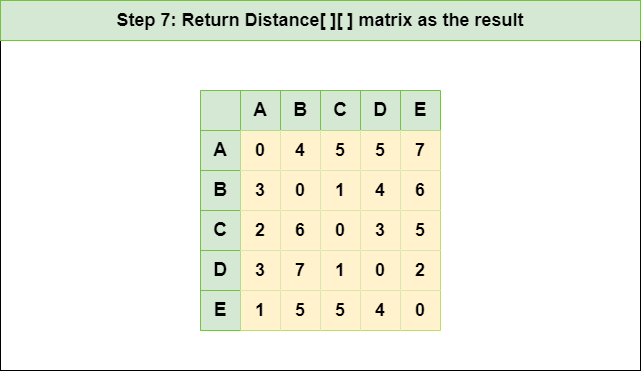
**

***Step 6****: Treat node* ***E*** *as an intermediate node and calculate the Distance[][] for every {i,j} node pair using the formula:*

*= Distance[i][j] = minimum (Distance[i][j], (Distance from i to* ***E****) + (Distance from* ***E*** *to j ))  
= Distance[i][j] = minimum (Distance[i][j], Distance[i][****E****] + Distance[****E****][j])*

**

***Step 7****: Since all the nodes have been treated as an intermediate node, we can now return the updated Distance[][] matrix as our answer matrix.*

**